Laboratory 6

Deadline: 2 week – Groups of 2 students

Part I

Write a simple image processing application in C/C++.

- The input to the application should be a *ppm* image file. There are many free utilities to convert from *jpeg* images to *ppm* ones.
- Your application must read this file and store the pixel information in a matrix.
- Then, it must perform two transformations to the image, one after the other. Choose some simple transformations such as “RGB to grayscale”, “edge detection”, “image blur”, etc. Let us call the two transformations *T1* and *T2*.
- Write the resultant pixel matrix to a new *ppm* file.
- Usage: ./a.out <path-to-original-image> <path-to-transformed-image>

Part II

Now suppose you have a processor with two cores. You want your application to finish faster. You can do this by having the file read and *T1* done on the first core, passing the transformed pixels to the other core, where *T2* is performed on them, and then written to the output image file. Do this in the following ways:

1. *T1* and *T2* are performed by 2 different threads of the same process. They communicate through the process’ address space itself.
    a. Synchronization using atomic operations
    b. Synchronization using semaphores
2. *T1* and *T2* are performed by 2 different processes that communicate via shared memory. Synchronization using semaphores.(Single source file)
3. *T1* and *T2* are performed by 2 different processes that communicate via pipes. (Single source file)

- Briefly describe the chosen image transformations in your report.
- Devise a method to prove in each case that the pixels were received as sent, in the sent order. Describe the method in your report.
- Try with three different image sizes.
- Study the run-time and speed-up of each of the approaches and discuss.
- Discuss the relative ease/ difficulty of implementing/ debugging each approach.

Submit a single zip file with the source code, a makefile, an input *ppm* image, and a report.

- *make part1* should compile the Part I version of the code and run it, creating the file *output_part1.ppm*
- *make part2_1a* should compile the multi-thread, atomic operation version of the code and run it, creating the file *output_part2_1a.ppm*
- *make part2_1b* should compile the multi-thread, semaphore version of the code and run it, creating the file *output_part2_1b.ppm*
- *make part2_2* should compile the shared memory version of the code and run it, creating

the file *output_part2_2.ppm*

- *make part2_3* should compile the pipe version of the code and run it, creating the file *output_part2_3.ppm*

Table format:

Image size	Part1(Sequential)	Part2.1a (Threads - atomic operations)	Part2.1b (Threads – Semaphores)	Part 2.2 (Process – Shared memory)	Part 2.3 (Process – Pipe)